Endian Issues

By supporting two memory-addressing modes, the PowerPC can run any OS or application

William Stallings

One of the annoying but important differences among processors is the way they store data in memory. Most processors use one of two data-organization strategies, known as big-endian and little-endian, which are described in detail below. (The term *endian* is derived from a passage in Jonathan Swift's *Gulliver's Travels* .) Some machines, such as VAXes and systems based on the Intel x86 or the Pentium, are little-endian machines; others, such as the IBM System 370, machines based on the Motorola 680x0, and most RISC machines, are big-endian.

The differences between these strategies are relatively minor in terms of performance and efficiency. However, programmers and users alike need to be aware of endianness, because data ordered in one format isn't compatible with data ordered in the other. This isn't a problem when dissimilar platforms operate autonomously. But in a networked environment that encourages program portability and data interchange across platforms, this can create problems.

Byte-Ordering

Endianness has to do with the byte-ordering of multibyte scalar values. The concept arises when it becomes necessary to treat a multiple-byte entity as a single data item with a single address, even though it's composed of smaller, addressable units. When a programmer assumes a specific endian format and attempts to manipulate the individual bytes or bits within a range of multibyte scalar values, problems can occur.

The following description of endian byte-ordering illustrates such a dilemma. Suppose you have the 32-bit hexadecimal value 12345678 stored as a 32-bit word in byte-addressable memory at byte location 184 . The value consists of 4 bytes, with the least significant byte containing the value 78 and the most significant byte containing the value 12. There are two ways to store this value: Start with value 12 in location 184, or start with value 78 in location 184.

The first mapping stores the most significant byte in the lowest numeric byte address; this is known as *big-endian* format. The second mapping stores the least significant byte in the lowest numeric byte address; this is called *little-endian* format. For a given multibyte scalar value, big- and little-endian formats are byte-reversed mappings of each other. In any machine, data aggregates such as files, structures, and arrays are composed of multiple data units, each with

endianness. Thus, the conversion of a memory block from one style of endianness to the other requires knowledge of the data structure.

The figure "Three Memory Orders of Structure K" illustrates how endianness determines addressing and byte order. The structure in the listing "A Multibyte C Data Structure" contains several data types. The memory layout in part (a) of the figure results from compilation of that structure for a big-endian machine; part (b) shows the results from compilation for a little-endian machine. In each case, memory is treated as a series of 64-bit blocks.

Several observations about this data structure can be made:

```
-- Each data item has the same address in both big- and little-endian
   schemes. For example, the address of the doubleword that has the
   hexadecimal value 545512134748BEBF is 08.

-- Within any given multibyte scalar value, the ordering of bytes in
   the little-endian structure is the reverse of that for the
   big-endian structure.

-- Endianness does not affect the ordering of data items within a
   structure. Thus, the four-character word
x3
 in the listing exhibits
   byte reversal, but the seven-character byte array
x4
 does not. H
ence,
   the address of each individual element of
x4
 is the same in both
   structures.
```

PowerPC Addressing Modes

The PowerPC is a *bi-endian* processor; that is, it supports both big- and little-endian addressing modes. This bi-endian architecture enables software developers to choose either mode when migrating OSes and applications from other machines. The OS establishes the endian mode in which processes execute; the default mode is big-endian. Once a mode is selected, all subsequent memory loads and stores are determined by the memory-addressing model of that mode.

To support this hardware feature, 2 bits in the MSR (machine state register) are maintained by the OS as part of the process state. One bit (ILE) specifies the endian mode in which the kernel runs when processing an interrupt; the other (LE) specifies the processor's current operating mode. Thus, the mode can be changed on a per-process basis, which is critically important for forei gn OS emulation.

When an interrupt occurs, the processor saves the current MSR and loads an MSR for the interrupt-processing routine. The value of the ILE bit in the old MSR is copied into the LE bit in the new MSR. When execution resumes in the interrupted process, its MSR is reloaded with its LE and ILE bits intact.

Byte Storage

The PowerPC architecture specification does not dictate how a processor should implement little-endian mode. It specifies only the view of memory that a processor has when operating in little-endian mode. When converting a data structure from big- to little-endian, the processor can either implement a true byte-swapping mechanism or use some sort of an address-modification mechanism. Current PowerPC processors are all big-endian by default and use address modification to treat data as little-endian.

Part (c) of the figure "Three Memory Orders of Structure K" shows how memory is laid out when data is stored in litt le-endian form for current PowerPCs. This is not a true little-endian organization as it is usually defined. Rather, it is designed to minimize the data manipulation required to convert from one endian format to another.

Note that 64-bit scalars are stored in the same formats on the PowerPC. To accommodate smaller scalars, a technique known as *address munging* is used. When the PowerPC is in little-endian mode, it transforms the 3 low-order bits of an effective address during a memory access. These 3 bits are XORed with a value that depends on the transfer size: 0x100 for 4-byte transfers, 0x110 for 2-byte transfers, and 0x111 for 1-byte transfers. The table "PowerPC Address Munging" lists the possible combinations.

For example, the 2-byte value 0619 is stored at location 1C in big-endian mode. In little-endian mode, it's viewed by the processor as still being stored in location 1C, but in little-endian mode. In fact, the value is still stored in big-endian mode, but at location 1A. When a transfer occurs, the system must do an address unmunging and a byte transfer to convert data to the form expected by the processor. The processor generates effective addresses of 1C and 1D for the 2 bytes. These addresses are munged (XOR with 110) to 1A and 1B. The data bytes are retrieved, swapped, and presented as if they were found in the unmunged addresses 1D and 1C.

Unaligned Data

This address-munging technique does not work cleanly with data that is not aligned on its natural boundary (e.g., a 4-byte value is aligned if its address is divisible by 4). When a value is unaligned, its storage in little-endian mode might result in the value being split into two noncontiguous parts. When an unaligned access is attempted in little-endian mode, an alignment interrupt occurs. This causes the processor to transfer to the system-alignment error handler, which handles the interrupt by a series of load-and-store operations that emulate the memory access.

Because of the exception processing, accessing unaligned little-endian data can seriously degrade a processor's performance. The simplest fix is to properly align little-endian data. But this might not be possible for certain processes, such as an x86 emulator, which accesses variable-length x86 instructions in memory.

But another solution is in the works. New versions of the PowerPC 603 and 604 will handle misaligned little-endian accesses in hardware, and thus handle an alignment interrupt the same way as in big-endian mode. They will be able to operate in little-endian mode without incurring a performance penalty.

Implications

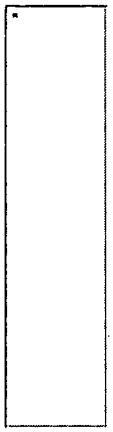
The PowerPC architecture is organized for big-endian storage and processing. It also provides a transparent method for dealing with little-endian programs and data.

This enables a PowerPC processor to run a program written for little-endian memory organization simply by recompiling the application on the PowerPC, which reduces the progr am-porting work required. When the recompiled program is run on the PowerPC with the LE bit set, the processor's address-mapping facility makes all data structures appear identical to the layout that the program saw on a little-endian machine. This ability to handle bi-endian address modes makes the

PowerPC processor ideal for hosting different OSes, such as on the CHRP (Common Hardware Reference Platform).

Three Memory Orders of Structure K

illustration link (15 Kbytes)



(a) Big-endian ordering of data.
(b) Little-endian ordering as seen by a PowerPC processor (true little-endian ordering).
(c) Little-endian ordering as found in PowerPC storage to minimize data swapping during memory accesses.

A Multibyte C Data Structure

illustration link (3 Kbytes)

PowerPC Address Munging

illustration link (5 Kbytes)

William Stallings is an independent consultant. This article is based on material from his most recent book, Computer Organization and Architecture: Designing for Performance, Fourth Edition (Prentice-Hall, 1995). You can reach him on the Internet at stallings@acm.org or on BIX c/o "editors."
